

Knowledge-Based Synthesis of Distributed Systems Using Event Structures

Mark Bickford, Robert Constable, Joseph Halpern, and Sabina
Petride

Cornell University

Program Synthesis

Premise: the goal of a distributed application is formalized as a **specification**, i.e. formula in a logical language.

Synthesis: given a specification, find a program that satisfies it.

Ideally

- extracted programs are close to *running code*
- specifications are written in a *high-level language*

Knowledge-Based Programs And Specifications

Knowledge is a natural and useful abstraction for formalizing specifications about distributed systems.

Example

The goal of message communication over a lossy channel is for the receiver to **know** the message and for the sender to **know** that the receiver **knows** the message.

Many distributed problems have natural formalizations in terms of knowledge

- distributed commitment (Mazer, Lochovsky '90)
- Byzantine agreement (Dwork, Moses '90; Halpern, Moses, Waarts '01)

Knowledge-Based Programs And Specifications

Agents act based on what they know or do not know

- more formally, preconditions on actions are **knowledge tests**

This intuition was formalized by Fagin, Halpern, Moses and Vardi ('95) as **knowledge-based programs**

- agent i knows fact φ written as $K_i\varphi$

Example

Agent i sends a message msg to agent j if he does not know that j already has the message, and does nothing otherwise:

```
if  $\neg K_i(has_j(msg))$  then send( $msg$ ) else skip
```

Typed language based on **computational type theory**

- **proving** $\exists x\varphi$ means **constructing** x such that φ holds

Higher-order logic

- possible to define abstractions, **knowledge** in particular

Proof assistant with an extensive library of proofs and tactics

Successfully used to **synthesize programs from specifications**

- if the spec has the form $\exists x\varphi$, then the constructed x is a program
 - ▶ Bickford, Kreitz, Renesse '01, Bickford and Constable '03

Program synthesis from kb-specifications using Nuprl

- focus on a fragment of Nuprl type-theory, called **event theory**
 - ▶ suitable for reasoning about executions of distributed systems
- add knowledge to the theory
- case study: the sequence transmission problem

We are interested in using a system to synthesize programs

Previous theoretical work

- deriving kb-programs from temporal kb-specs, assuming synchrony or only one agent-systems
 - ▶ Engelhardt, van der Meyden, Moses '00 - '01
- deriving protocols from kb-specs
 - ▶ van der Meyden '96

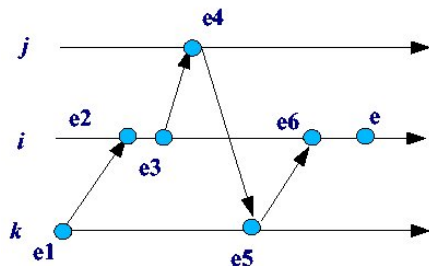
Event Structures

The theory of event structures in Nuprl is a formalization of Lamport's model of distributed systems:

- changes in the local state of an agent are modeled as **events**
- three kinds of events
 - ▶ *local*: the agent (nondeterministically) chooses a value for a variable
 - ▶ *send*
 - ▶ *receive*
- local and send events have associated a **value**
 - ▶ message sent / value chosen

Causal Order

- every *receive* event e has a corresponding *send* event $\text{send}(e)$
- $\text{send}(e)$ *causally* precedes e
- events associated with a particular agent are *totally ordered*



Causal order: transitive closure of the sender-receiver and local order relations

Distributed Message Automata

A **message automaton** A for some agent i is a formalization in Nuprl of a nondeterministic state machine for i

- specifies when and what actions i can take
- built from a small set of **basic clauses** which
 - ▶ define the initial value of one state variable
 - ▶ define the effect of some event on a state variable
 - ▶ define the preconditions for local events to occur
 - ▶ restrict the set of events that result in changing a variable's value
- more complex automata formed by composition
 - ▶ $A \oplus B$ is the union of clauses of A and B

A **distributed message automaton** is a collection of message automata, one for each agent.

To each **basic clause** corresponds a **basic rule** in Nuprl that characterizes the specification satisfied by the clause.

Example

Basic clause: $@i \ a(v) \text{ only if } P(v)$

Basic rule: an event labeled a with value v occurs only if precondition P holds in the current state of i .

Example

Basic clause: $@i \quad a(v) \text{ sends } f(v) \text{ on } l$

Basic rule: whenever an event e with value v occurs, the message $f(v)$ at the current state of i is sent on link l .

Example

Basic clause: $@i \quad \text{only } a \text{ events send on } l$

Basic rule: only events labeled a result in i sending on link l .

More Complex Rules

If A satisfies φ_1 and B satisfies φ_2 , then $A \oplus B$ satisfies $\varphi_1 \wedge \varphi_2$.

Example

Fairness specification $\text{Fair}_i(P, f, l)$: conjunction of a safety and liveness condition

- i only sends messages on l when P holds, and then sends f 's value
- infinitely often, either a message is received over link l or P fails
 - ▶ typical assumption in distributed systems

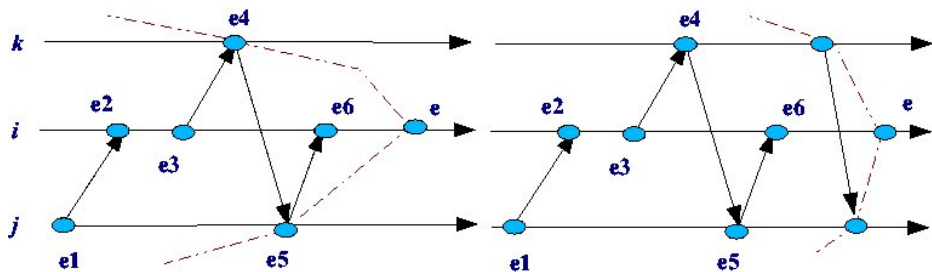
$\text{Fair}_i(P, f, l)$ is satisfied by $\text{Fair-Pg}_i(P, f, l)$

@ i $a(v)$ only if $P(v) \oplus$

@ i $a(v)$ sends $f(v)$ on $l \oplus$

@ i only a events send on l

Adding Knowledge To Nuprl



An agent is uncertain about what events have occurred.

- agent *i* considers possible any sequence of events that is causally consistent with the events *i* has recorded so far
- *i* knows a fact φ if φ is true in all global configurations *i* considers possible
 - ▶ natural definition of knowledge, which can be expressed in Nuprl

$\text{Fair}_i(P, f, l)$ can be generalized to a fairness condition $\text{Fair}_i(K_i\varphi, f, l)$ involving knowledge

- i only sends messages on l when he knows φ , and then sends f 's value, and
- infinitely often, either a message is received on link l or i does not know φ
 - ▶ $K_i\varphi$ is a *kb-predicate*; it takes as an argument an event structure

$\text{Fair}_i^{kb}(K_i\varphi, f, l)$ is satisfied by the kb-program $\text{Fair-Pg}_i^{kb}(K_i\varphi, f, l)$

The sequence transmission problem

STP problem: one agent (the sender) has an input tape $X = \langle x_1, x_2, \dots \rangle$ of data elements that he wishes to communicate in order to a second agent (the receiver).

- Many protocols are known that solve STP:
 - ▶ Stenning's protocol
 - ▶ the alternating bit protocol
 - ▶ Aho, Ullman and Yannakakis's solution

Kb-specification for STP: eventually the receiver knows X

- formal spec $\varphi^{kb} \equiv \diamond \forall i K_R(X[i])$
- a simple kb-program satisfies φ^{kb} (Halpern and Zuck)
- all programs above implement it

Our goal: derive this kb-program from the kb-spec, using Nuprl.

Synthesizing A Kb-Program For STP

Idea: achieve φ^{kb} by ensuring R makes progress.

- for each i , if for all $j < i$ there was a time when R knew $X[j]$, then eventually R knows $X[i]$

Need to assume a *fairness condition*

- all messages sent infinitely often are eventually received
 - ▶ otherwise, STP cannot be solved

Synthesizing A Kb-Program For STP

How can S ensure that R learns the i th bit?

- infinitely often either S sends $X[i]$ or S knows R knew $X[i]$ at some time in the past
 - ▶ instantiate Fair_S^{kb} using $P_S^{kb} \equiv \exists i \neg K_S(\diamond K_R(X[i]))$ and function f_S that returns $\langle i, X[i] \rangle$
 - ★ call the spec Fair_S^{kb}

How does S learn which bits R knows?

- R sends a request for $X[i]$ only if he knows $X[j]$ for all $j < i$, but does not know $X[i]$

How long should R send a request for $X[i]$?

- as long as R does not know $X[i]$
 - ▶ again, instantiate Fair_R^{kb} for $P_R^{kb} \equiv \exists i \neg \diamond K_R(X[i])$
 - ★ call the spec Fair_R^{kb}

Synthesizing A Kb-Program For STP

Use the system to prove

- if both Fair_S^{kb} and Fair_R^{kb} satisfiable, then φ^{kb} satisfiable

Do this by instantiating the general proof that Fair is satisfiable

- $\text{Fair-Pg}_S^{kb} \models \text{Fair}_S^{kb}$ and $\text{Fair-Pg}_R^{kb} \models \text{Fair}_R^{kb}$

Then compose the two kb-programs

- $\text{Fair-Pg}_S^{kb} \oplus \text{Fair-Pg}_R^{kb}$

$\text{Fair-Pg}_S^{kb} \oplus \text{Fair-Pg}_R^{kb}$ satisfies the spec!

- similar to the program proposed by Halpern and Zuck

Synthesizing A Standard Program

We have now derived a kb-program for STP, but we really want a standard program.

- The extracted program has kb-tests

$$\text{@S } a(i) \text{ only if } K_S(\diamond K_R(X[0])) \wedge \dots \wedge K_S(\diamond K_R(X[i-1])) \wedge \\ \wedge \neg K_S(\diamond K_R(X[i])) \oplus \dots$$

- However, the proof is correct even for predicates $K'_S\varphi$ and $K'_R\varphi$ that do not have all the properties of knowledge

Goals:

- identify which properties are necessary for the proof to succeed
- use this insight to replace the kb-tests by standard tests

Synthesizing A Standard Program

Replace

- $K_S(\diamond K_R(X[i] = v))$ by an abstract predicate $P(X[i] = v)$
- $K_R(X[i] = v)$ by $Q(X[i] = v)$

Define $\tilde{\text{Fair}}_S^{kb} \oplus \tilde{\text{Fair}}_R^{kb}$:

@S $a(i)$ only if $P(X[0]) \wedge \dots \wedge P(X[i-1]) \wedge \neg P(X[i]) \oplus \dots$

$\tilde{\text{Fair}}_S^{kb} \oplus \tilde{\text{Fair}}_R^{kb} \models \varphi^{kb}$ if

- P and Q are **sound tests**
 - ▶ $P(X[k] = v) \Rightarrow (X[k] = v)$ and $Q(X[k] = v) \Rightarrow (X[k] = v)$
- after the event of R receiving $\langle i, v \rangle$, $Q(X[i] = v)$ holds
- after the event of S receiving i , $\forall j < i P(X[j])$ holds
- P and Q are **stable**

By taking P and Q to be standard, we get a standard program satisfying the spec.

Synthesizing Stenning's Protocol

Assume:

- S keeps track of the smallest i he has not received from R in i_S
- R keeps track of the smallest i such that he has not received $X[i]$ from S

Take $P(X[i]) \equiv i_S > i$ and $Q(X[i]) \equiv i_R > i$.

- P and Q satisfy the identified constraints
- the corresponding program $\tilde{\text{Fair}}_S^{kb} \oplus \tilde{\text{Fair}}_R^{kb}$ is standard (no knowledge tests) and corresponds to **Stenning's protocol**

Conclusion

- Synthesis of programs from kb-specs can be carried out in Nuprl
 - ▶ natural definition of knowledge in event theory
 - ▶ natural generalization of existing general specs to kb-specs
 - ▶ most of the proofs is automatic
 - ▶ synthesized programs are close to running code
 - ★ work in progress on translating these automata to Java code
- The proof for STP suggests an approach for treating synthesis of kb-programs and of standard programs as successive stages of the same process